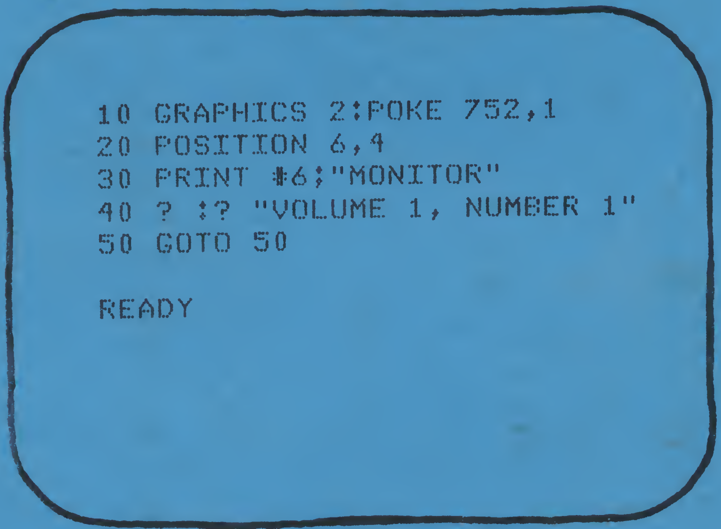
$ 2.00

# MONITOR

**a journal for Atari® users**



```
10 GRAPHICS 2:POKE 752,1
20 POSITION 6,4
30 PRINT #6;"MONITOR"
40 ? :? "VOLUME 1, NUMBER 1"
50 GOTO 50

READY
```

**volume 1, number 1**

# MONITOR

a journal for Atari® users

################################################################################

## IN THIS ISSUE

PUBLISHERS/EDITORS

Karen Leavitt
Frieda Lekkerkerker

CONTRIBUTING WRITERS

Bill Blocher
Dave Hallowell
Richard Kaye
Peter Swiggart

PRINTING

Bruce Brown

P.O. BOX 365
Newton Highlands,
MA 02161

## WELCOME FROM THE EDITORS!

MONITOR is a journal that aims to serve owners of Atari computers with information about products and to support them with helpful programming hints. Computing has become a way of life for a great many people; it is a way, however, that does not sustain itself, but needs and enjoys the activity of learning and teaching. The number of computer magazines that have come on the market recently and their enormous popularity should tell us something about this need and enjoyment. Atari users may find resourceful information in a variety of these magazines - you will find a list in our "Recommended Reading" - but, as we learn from users in the area, many questions remain unanswered and some expectations unfulfilled. Our intention with this journal is to supply answers to simple as well as complex questions. More than anything, we want to keep in mind that what is simple to an experienced user may seem terribly complex to a beginner.

Some of the contributors to the MONITOR are computer professionals, others are experts in fields that have become related to computers; all share a great enthusiasm for the Atari and a wish to explore the computer to its fullest potentials. We hope that you will find this enthusiasm reflected in this publication and will benefit from the results of our explorations.

We will review software and hardware for you in a detailed and critical way. You will see articles on programming with actual examples to serve as a basis for your own computing endeavors. REM statements and annotations will be plentiful to make the BASIC code as clear as possible. We also expect to publish Assembly routines and to give advice on issues concerning the system. There is a column that lists reading material for the beginning as well as the advanced user.

We intend to publish MONITOR on a bi-monthly basis. If you have questions to ask or contributions to make, please contact us at P.O. Box 365, Newton Highlands, MA 02161. We are looking forward to receiving your response!

[illegible]

[illegible]

# Under The Hood of Your Atari Computer

by Bill Blocher

If you are the lucky owner of an Atari home computer, you can pride yourself on having a keen sense of computer hardware. You have in your possesion the hardware bargain of the century. The architecture is "state of the art." No other home computer can boast of the capabilities of the Atari.

There are those who try to compare Apple, Radio Shack, Zenith and all others with the Atari. It is only sheer ignorance of the the Atari hardware that makes them do so. Comparing Apple and Atari is like comparing a bicycle to the Space Shuttle. All other personal computers have one processor, which is the "host" and, with the exception of a possible quasi-processor to generate video output, the only processor. The Atari has a four-processor network! This computer network, allows capabilities that other computer owners can only dream of.

The host processor is a 6502 model B. Its speed is 1.79 megahertz. Other 6502-based computers such as Apple have the 6502 model A processor with a speed of 1 megahertz. This means that a machine language program written on the Atari will run about twice as fast (assuming the displays are turned off). The host processor is where any architectural similarities stop. Another processor under the hood of the Atari is the "Antic" processor. Antic is a special graphics work-horse which controls the screen display. It has its own instruction set which allows the mixing of graphics modes on the screen. A program for Antic is referred to as a "Display List". Antic controls, smooth horizontal and vertical scrolling of the screen. It also supports a feature called "Display List Interrupts," This is an extremely powerful feature. Without getting too technical, it is mechanism which allows specialialized processing at times when the television electron beam is at selected vertical positions. One could talk about the features of the Antic for days. The fast 6502 processor and the Antic processor together form the "Dynamic Duo," a team of super processors which is the start of a new generation of super machines.

The next super hero in the Atari, is the CTIA processor (GTIA in machines built after December 1981). This processor controls the color of the TV screen, Player-Missile Graphics (with the help of ANTIC), collision detection, horizontal smooth scroll of players and misiles, and more!

Player-missile graphics is an outstanding feature of Atari computers. It allows the programmer to design four (it is possible to have five) independent objects which can have priority over other objects and be moved independently on the screen. Missiles are objects like players, but smaller in width; they make great bullets or can be ganged together to form a fifth player. Collision detection, mentioned earlier, is the ability to detect (in hardware) whether a player-missile has touched another player or backround.

Next, let me introduce you to Mr. Pokey, the fourth processor (let's now call them the fantastic four) in the Atari computer system. Pokey is the sound chip; it has four semi-independent audio channels. One can control the tone, pitch and volume of each channel. Anything from space noises, bombs dropping to music can be generated.

Besides sound, Pokey controls the paddle controllers, the serial port control, random number generator, and system interrupt control.

There is also a fifth chip inside the Atari called a PIA (a common chip found many computers). The PIA controls the joystick ports and the cassette interface. The joystick ports are really four bits of parallel input or output (one can select the direction). This allows the development of devices to plug directly into the joystick ports. An application we are working on at The Bit Bucket is a robot. We use a sonar device for the robot's eye, which we plug into two joystick ports.

When you add the architectural features not mentioned here (22 address lines, vector-oriented operating system ....) you wind up with an incredible microprocessor. Let those, who will, try to compare the Atari with other home computers. I compare the Atari to $10,000 - $15,000 graphic processors.

# BASIC TUTORIAL:

## Introduction to Graphics Modes

by Karen Leavitt

As most Atari owners already know, we have nine different graphics modes in which we can produce displays on the television screen.

TEXT MODES

Of these, three, modes 0,1, and 2, are text modes. That is, you cannot draw lines and designs in modes 0,1, and 2, you can simply display alphabetic and numeric characters in various sizes. In fact, Atari's default mode (the mode the Atari selects unless you tell it otherwise) is mode 0. This is the familiar blue screen with white letters (40 characters per line, 24 lines per screen) you see when you turn the power on. This mode is fine for typing in BASIC programs, but as far as producing eye-catching displays is concerned, it's pretty dull.

To use the other two text modes, you must tell the Atari (via instructions in BASIC) to switch to mode 1 or mode 2.

To do this, type:

```
GRAPHICS 1
  or
GRAPHICS 2
```

Notice that most of the screen is black, but there is a blue space at the bottom. This is known as "split screen" mode. The blue area is a "text window" in which you can type instructions in BASIC. If you want to eliminate the text window, add 16 to the number of the graphics mode you choose.

e.g., GRAPHICS 1+16 or GRAPHICS 17

In text mode 1, characters will be the same height as in mode 0, but they will be double the width. Hence in this mode you still have 24 lines of text, but now there are only 20 characters on a line. Actually, because the blue window at the bottom of the screen occupies four lines, you only have 20 lines.

As in text mode 1, the characters in mode 2 are double width (20 per line), but these are also double height. So we now have just 12 lines of text (10 lines if you use a split screen).

USING TEXT MODES

To make use of modes 1 and 2, simply type the BASIC instruction GRAPHICS followed by the number of the mode you want (plus 16 if you don't want a split screen). The GRAPHICS instruction may be used in either deferred or immediate instruction mode (that is, with or without a line number).

The last thing you must know before you can print enlarged text on the screen is that you must specify a "device code" in your print statement. That's not really as confusing as it sounds; if you just execute a PRINT statement the Atari thinks you want to write in the blue text window at the bottom of the screen. Fortunately when you issue a GRAPHICS command, the Atari assigns a number to the large blank screen so you can identify it in your PRINT statement. This number is always 6. So to print text on the screen in mode 1 or 2, type:

```
PRINT#6;"message"
```

where "message" is what you want to appear on the screen.

Here is a sample program you can type in that will fill up a graphics mode 2 screen by numbering each line. How would you modify it to use the whole screen instead of a split screen? How about changing it to work in mode 1?

```
10 REM COUNTING PROGRAM FOR GRAPHICS MODE 2
20 GRAPHICS 2
30 FOR LINENUM=1 TO 10
40 PRINT#6;"THIS IS LINE ";LINENUM
```

```
50 NEXT LINENUM
60 GOTO 60 : REM INFINITE LOOP
```

Line 60 of this program tells the computer to go into an infinite loop. This will keep the program running until you hit the [BREAK] key or the [SYSTEM RESET] key. You will see why this line is necessary if you try using mode 2+16 without it.

Note that I've called the loop variable "LINENUM." Atari BASIC allows us to use very long variable names, and it is good programming practice to give informative names to variables so that someone reading your program can understand it easily. It's also a good idea to use REM statements whenever you have a particularly cryptic piece of code.

## GRAPHICS MODES

Now that you've experimented with text modes, you're probably anxious to make your Atari display some more interesting pictures. For this we'll use graphics modes 3 through 8.

In these modes we don't print text on the screen; instead we use special graphics instructions to display dots on the screen. We can position these anywhere we want, and mix the dots together to form shapes.

This is beginning to get interesting, but why do we need six different graphics modes just to lump together a bunch of dots? Just as the different text modes provide characters of varying sizes, the different graphics modes have dots of different sizes. These dots are called pixels--a contraction of "picture elements"--because they can be combined to form a larger picture. In these modes we speak of "resolution" to refer to the number of pixels that can fit onto a screen. The fewer the pixels, the lower the resolution, and the more pixels on the screen, the higher the resolution.

Graphics mode 3 is the lowest resolution mode with a maximum of 40 pixels across and 24 down (20 with split screen). Modes 4 and 5 offer 80 pixels horizontally and 48 (40 split) vertically. 6 and 7 have a resolution of 160 x 96 (80), and mode 8 is the highest resolution mode with 320 x 192 (160) pixels.

## USING THE GRAPHICS MODES

Just as with the text modes, we invoke Atari graphics modes by using the GRAPHICS instruction.

e.g., GRAPHICS 6 or GRAPHICS 4+16

In order to tell the Atari which pixels you want to light up you use the PLOT instruction. You simply type PLOT followed by the X and Y coordinates of the pixel you want to illuminate. X is the horizontal position, Y is the vertical. Remember that pixels are labeled from 0 to max-1, so if you are in mode 6, the first horizontal position is 0, and the last is 159.

Here is a little program to draw a line using the PLOT statement:

```
10 REM A LOOP TO PLOT CONTIGUOUS HORIZONTAL POINTS IN MODE 6
20 GRAPHICS 6
30 COLOR 1 : REM TO BE EXPLAINED LATER
40 YCOORD=25
50 FOR XCOORD=0 TO 159
60 PLOT XCOORD,YCOORD
70 NEXT XCOORD
80 GOTO 80
```

If you typed this in and executed it, an orange horizontal line should have appeared on your screen. The COLOR instruction in line 30 will be discussed later, but see what happens if you don't use it.

If we want a diagonal line, all we have to do is increment the Y coordinate as well as the X coordinate. Run that last program again, but this time include the line:

```
55 YCOORD=INT(XCOORD/2)
```

This will increment YCOORD at half the rate of XCOORD because the vertical resolution in mode 6 is only half the horizontal resolution.

Going through a loop to plot points in a line can be tedious, so Atari has provided another instruction that takes the drudgery out of line-drawing. It's called DRAWTO. The format is:

```
DRAWTO x,y
```

It looks a lot like a PLOT instruction, doesn't it? What it does is draw a line from the last pixel lit up on the screen to the point specified by x and y. For example, the program we just did to draw a diagonal line from upper left corner to lower right corner could be simplified as follows:

```
10 REM DRAW A DIAGONAL LINE USING DRAWTO
20 GRAPHICS 6
30 COLOR 1
40 PLOT 0,0
50 DRAWTO 159,79
60 GOTO 60
```

That was quite a bit faster, too, wasn't it?

Great. We now know how to display text and draw lines all over the screen, but you've probably noticed that everything is orange. What about all the colors Atari's supposed to have? Well, here we go.

## COLOR

### COLOR IN GRAPHICS MODES

Using color in Atari graphics modes is a twofold process. It can be thought of as follows:

Step 1: Fill a bucket with paint

Step 2: Dip your brush into that bucket

We may think of a third step (that of stroking the brush on our "canvas") as translating into a PLOT or DRAWTO statement. After some more discussion of the graphics modes themselves, we will examine those first two steps.

By now you've no doubt noticed that some of the graphics modes are not unique in their resolution. The difference is the number of colors available in each mode. Mode 3 has four colors (including background color), so do modes 5 and 7. Modes 4 and 6 have two colors. Mode 8 has just one color with two different intensities, or luminances. Let's discuss the 4-color modes first.

Officially, the buckets are called "color registers." In modes 3,5, and 7 our buckets (registers) are labeled 1, 2, 3, and 0. Register 0 determines the color of the canvas; that is, this register contains the background color. Registers 1, 2, and 3 contain "paints" to be used as foreground colors. The way you tell the Atari which bucket you want to paint from is via the COLOR instruction. It's format is:

```
COLOR bucket#
```

Where bucket# is, of course, the number of the bucket into which you want to dip your brush. Every PLOT statement issued after the COLOR instruction will be executed using paint from that COLOR statement.

Here is a program that will draw three different-colored lines on a black background in mode 7:

```
10 REM EXAMPLE OF COLOR INSTRUCTION
20 GRAPHICS 7
30 FOR Y=1 TO 3
40 BUCKET=Y
50 COLOR BUCKET
```

```
60 PLOT 0,Y*15
70 DRAWTO 159,Y*15
80 NEXT Y
```

You should have an orange line, a green line, and a blue line on your screen. Well, what if those weren't the colors you wanted? How do we empty the buckets and fill them with a different color paint? To do this, we must use the SETCOLOR instruction. Here is the format of the SETCOLOR instruction:

```
SETCOLOR register,color,luminance
```

The color part of the instruction must be a number from 0 to 15. (The color values for these numbers are listed in fig. 1, page 28.) This is the color that is put into the register, or bucket, specified as the first argument. The luminance is the intensity, or shade, of the color you want. This must be an even number from 0 to 14. 0 is the darkest shade--close to black, 14 is almost white. By inserting this color and luminance information into the bucket specified, you automatically change the color of every line drawn using paint from this bucket.

This all probably seems very straightforward, and it is. Except for one problem--you don't use the same bucket numbers in the SETCOLOR instruction as you do in the COLOR instruction. (you can blame this on the Atari engineers) So, in graphics mode 7 if you plot a point using color register 1,

e.g.,

```
10 COLOR 1
20 PLOT x,y
```

to change the color in that same bucket you must say

```
SETCOLOR 0,color,luminance
```

A complete chart of COLOR/SETCOLOR correspondence is shown in fig. 2, page 28. Although the discrepancy in the register numbers in the COLOR and SETCOLOR statements can be very confusing, with practice, you should be able to overcome this difficulty.

## <u>COLOR IN TEXT MODES</u>

To change colors in modes 1 and 2, we don't use the COLOR instruction. Instead, our choice of characters determines the color that appears. If we print regular upper-case characters they will draw their color from SETCOLOR register 1. Lower-case characters will take color from SETCOLOR register 2. Inverse video (type the Atari logo key) upper-case characters look to SETCOLOR register 3, and inverse lower-case characters rely on SETCOLOR register 4. To illustrate this, type in the following program.

```
10 GRAPHICS 1
20 PRINT#6;"ONE"
30 PRINT#6;"two"
40 PRINT#6;"THREE" : REM INVERSE VIDEO CHARACTERS
50 PRINT#6;"four" : REM INVERSE
```

Run this program and notice that each word is in a different color. Now experiment with color changing by typing in random SETCOLOR statements while the text is still displayed on the screen. For example, at the READY prompt, type:

```
SETCOLOR 2,8,6
```

What happened? Keep typing in SETCOLOR statements, each time changing the register number or the color or the luminance until you feel comfortable with the operation of the SETCOLOR statement.

That's all for now on standard BASIC graphic modes and color. Next time we'll discuss three new graphics modes that have materialized because of the installation of a new chip, the GTIA chip, by Atari.

## REVIEW:

Santa Cruz Educational Software
5425 Jigger Dr.
Soquel, CA 95073
Tricky Tutorials #1-#4
$19.95 each, $64.95 set of four

The Atari owner who has advanced beyond the Novice level of simple Prints and Plots and who seeks information about such intriguing matters as Display List or Player Missiles may be in for a bit of frustration. The literature on these subjects is scattered and scarce. The Basic manual gives no solace and although quite a few magazine articles have appeared, they are often inconclusive - "more about this next month" - or based on material discussed in earlier issues. There is, of course, De Re Atari, which deals with the many aspects and potentials of the Atari in an extensive and comprehensive way. The problem there is that the text appears to have been written for the more advanced programmer who is familiar with Assembly Language or at least at ease with the various Peeks and Pokes. For those short of the Commander level the programs, offered by a company called Santa Cruz Educational Software, could be of some use. They come under the heading of "Tricky Tutorials" and consist of six programs, which can be bought separately or as a whole at a small discount. The subjects of these tutorials are, in order of their individual numbering: Display List, Horizontal/Vertical Scrolling, Page Flipping, Basics of Animation, Player Missile Graphics and Sound. I bought the first four programs with their promise to teach me some of the tricks of the Atari trade. The tutorials are clearly geared towards the user with no prior knowledge of all the tricky wizardry in the machine; the breezy tone of the manuals and the principle of allowing the user to modify programs and see the result almost instantly reflect this. For that reason alone, I find the enterprise worthy of some appreciation. There are drawbacks to this approach, however, as there are differences in quality and quantity between the various programs. To make my point clear, I propose to review each of the four programs individually.

"Display List" deals with the mixing of graphics modes on the screen: in other words, organizing the display list in such a way that the screen would show ,for example, one line of graphics mode 2, followed by a section of mode 8 and at the bottom a few lines in mode 0. The tutorial does not deal with D.L. interrupts , nor does it, in any real way, discuss those modes that cannot be accessed through Basic (infamous mode 7+, for example). In spite of these restrictions, I believe it gives ample and, above all, easy opportunity to customize display lists. There are 11 examples in this program and in each of them the user sees the actual list and is invited to make changes by moving the cursor around and typing in new numbers and/or deleting others (thanks to the editing power of the Atari!). Pressing return then shows the result of all the modifications. The manual provides charts to use with the modification and is helpful in calculating the numbers necessary for the custom screens. It explains about pixel lines and mode lines and emphasizes locations 87, 88 and 89 - important if you want to put anything on the screen. The most useful part of the program is example 10, which, after the user has changed the numbers in the D.L., indicates whether the new list has too few or too many lines, plus the advice to use the chart and make up the difference; then, it shows the numbers to poke in locations 87-89. Together with example 11 it should prepare the programmer to create his or her own list. All the examples can be used as subroutines in one's own programs, an additional helping hand of this tutorial. The only problem I have with "Display List" is that the helping hand never really lets go: in spite of its tutoring purpose, the general tone of the program is that you do not necessarily need to understand everything: just follow the directions and copy all the numbers for your own list. As a result the user may not be aware that those particular numbers are for his or her particular memory size, so that your neat display on your rich 48k machine will not run on your less fortunate friend's 32k. Yet, as a starting point to the understanding of display lists this tutorial will satisfy the curious mind. On to some scrolling.

Tutorial #2 deals with horizontal and vertical scrolling, of the coarse as well as the fine variety. There are many examples in this program: eighteen in all. If only the manual would be as long and plentiful as the number of listings! It begins with a short description of what a display list is and the function of LMS (load memory scan) - essential for doing any modification to make scrolling possible. But the information is short, as are the explanations that go with each example. The real value of this program lies in the opportunity for a close study of the listing of each example. Yet a true tutorial should be explicit about the function of the various statements in the programs. I found myself rushing through the examples and, naturally, getting lost in all the indeterminate scrolling motions. Particularly, in the section on fine scrolling, the educational aspect of the program loses sight of its goal. Why let the user poke the amount of "clock cycles" in a certain location only to add that the meaning of "clock cycles" will not be explained in the tutorial? All this with the reasoning that experimenting will teach more than paper descriptions! And why give an Assembly routine for vertical scrolling and none for horizontal, leaving the poor programmer with an all-Basic example, including, of course, those awful blinks? In all, the tutorial on scrolling can be helpful for the patient "lister," but if you expect to receive conclusive information about the subject you will not find it in this tutorial. The final remark of its author, "maybe I should have called this Introduction to Scrolling" seems quite appropriate. I truly hope that the manual can be rewritten and some thoughts given to the pedagogical side of tutoring. The subject of scrolling is very "tricky" and once mastered (with the help of Assembly Language) powerful and impressive: think of, for instance, "Eastern Front." But in order to understand, and explain, more is required than the size and quality of this tutorial.

Number three of the series is on page flipping. Ever since I read this sentence (from a catalog by the same software house), "Apples can't begin to do page flipping like this," I have been intrigued by the subject matter. Utter this sentence to any non-computer friend and you will get some strange looks! But on to the actual tutorial. It consists of eight examples, whose listings are all included in the manual. Two methods are used for flipping screens, one in which two or more complete display lists and display data are set up, and one that works with instant modification of the byte in the display list that indicates the beginning of screen memory. The manual, again, is not very informative about the techniques used. If you consider buying this program, it would certainly be advisable to do so after having gone through the Display List tutorial, or any other source of information.

Do not expect to see smooth and fast animation in the examples of this tutorial. The techniques used are too simple and basic for that purpose. But if you are satisfied with and keen on being able to change the screen with a simple keystroke, then the program may give you some insight and ideas in how to go about it. Are you anxious for some livelier work? Let us then look at the final tutorial.

"Basics of Animation" is true to its title: it teaches only the very "basic" principles of animation. The author begins his manual with the remark that animation cannot be done from Basic, but continues to console his readers that there are a few ways to get around this problem. Since his examples are, for the most part, written in Basic, and since buyers of this program will all expect to use his information in their Basic programs, this will raise quite a bit of expectation. However, for most users this consolation will be as disappointing as the images created by the examples of this tutorial. The program consists of three parts each of which describes a way to create some live action on your screen. The first two teach you how to use PLOT and PRINT commands in order to put an image on the screen, erase it through either the use of background color or that of preceding blanks and then PLOT or PRINT again in a new location, and so on. All this, as many readers will know, will not result in stunning animation. The only way in Basic for some faster action is through Player/Missiles and this is the subject of the third part of the tutorial. An eager user who might expect some instruction on this difficult topic will be very disappointed. P/M here appears only in the form of an inane game which will bore and infuriate everyone who sees it. It requires paddles, thus no vertical movement! The comments in the manual and REM statements in the program are minimal. True, the writer apologizes for this lack of information, adding that the subject is too difficult to be treated in an introductory lesson and advises to buy his Player/Missile Tutorial #5! Whatever the intentions of this kind of approach, this tutorial should have included more information on P/M, leaving out such unnecessary additions as the "Bird at the Ocean" from the ATARI BASIC manual and the ubiquitous ATARI logo with its 128 colors rolling down.

The computer is a warm and fascinating medium. The number of people who want to buy one and use it at home is growing enormously. The reason for this, I believe, is not just the presence of "Star Raiders" and others, but also the nature of the machine: its interactive, often friendly invitation to be manipulated. Once mastered, the possibilities for creativity are endless. The Atari, in particular, is very inviting, but the road to full understanding is hard. Santa Cruz Educational Software saw the need of so many people and has come out with these tutorials. Whether they will satisfy those needs is somewhat doubtful. In many ways these tutorials seem to be still unfinished; their author openly invites users to send in their comments and suggestions for improvement. One way that would certainly, to my mind, make the tutorials more attractive and instructive is to combine the first three of the series - they all deal with and are depended on an understanding of the display list - and add the better parts of "The Basics of Animation" to tutorial #5: "Player Missile Graphics" (not reviewed here). Then the user would have a more complete package in hand and there would no longer be a need for so much self-advertisement within the tutorials themselves! As it is now, the first tutorial, "Display List," combined with a memory map - available from Santa Cruz! - is complete and independent and will give you what its term indicates: instruction.

F.L.

Simulated Computer
EDU-SOFT
4639 Spruce St.
Philadelphia, Pa. 19139
$14.95 cassette; $19.95 disk

I have always contended that the thing a computer can teach best is how to use a computer. Now there is a program from Edu-Soft that simulates a computer and permits the user to see the way a computer operates.

The Simulated Computer, which requires BASIC and 32K, loads automatically when the computer is powered on. The first thing that appears on the screen is the title and directions telling the user to press the [SELECT] key in order to getoperating instructions, or the [START] key to bypass the instruction display and get right into the simulation. My first time out, I chose the [SELECT] key and immediately a command summary appeared on my screen. This list includes directives to the simulated computer to LOAD, RUN, clear memory (NEW), and single-step (RUNSTEP) through a program, as well as several other commands. I pressed the space bar and was presented with a brief explanation of the components of the simulates computer's instructions (each instruction is a three-digit number comprising a one-digit operation code, and a two-digit address code), and a summary of the ten possible op-codes. Pressing the space bar again revealed an explanation of the address code (a two-digit number from 00-19) and a summary of the STOP and SKIP instructions that the Simulated Computer (from now on referred to as SC) supports. One more tap on the space bar results in instructions on how to interrupt execution of the SC (type "B"), how to resume execution ("CONT"), and a summary of error codes--there are eight possible errors. Depressing the space bar once more brought me to the SC itself.

The command summary would mean nothing to a neophyte, so I followed the advice in the accompanying user's manual, and ignored much of what I read until I got to the part where the simulation began (pressing the [START] key in the first place would have taken me here).

For the simulation, twenty rectangles are drawn on the screen labeled from 00 to 19. Printed above these is the label, "MEMORY." At top part of the screen are two large boxes, one labeled "INPUT," the other "OUTPUT." Between these are three small rectangles which fall under the general heading "REGISTERS," and are individually labeled "AC," "PC," and "IR." At the very top of the screen is a notice reading "DISPLAY MODE ON"--I guess that means the display mode must be on.

As directed by my SC tutorial manual, I located the various components of the SC and entered a sample program into the SC. I began by typing the command LOAD in response to the "?" prompt in the INPUT box. The computer displayed the label of the first memory location and waited for me to enter the three-digit numbers provided in the tutorial manual. As I typed in each number and pressed [RETURN] the number appeared in the rectangle on the screen bearing the appropriate label--the computer was storing my instructions in memory!

I do have one complaint regarding this stage in my progress. Had I made a typing error at this point, I would  have been completely at a loss for how to correct it. Fortunately, I only had to contend with seven, three-digit numbers, and I made it through flawlessly.

After I'd successfully "LOADed" my program, I had to "RUN" it. Watching the program run was sheer delight. It was a simple infinite loop that incremented the accumulator by one each time and printed its contents. The progress of the program was charted by the box labeled "PC" (program counter) which contained the number of the next memory location to be processed. Meanwhile, the current memory location was highlighted and the current instruction appeared in the box labeled "IR" (instruction register). The contents of the box labeled "AC" (accumulator) were incremented whenever the appropriate instruction was executed, and whenever an output instruction was reached, the contents of the accumulator would appear in the box labeled output. Throughout all of this, the spot in which the "DISPLAY MODE ON" message had once resided was constantly changing to display a brief explanation of the instruction being executed.

I found that I could slow down the execution time of the SC by pulling back on the joystick connected to the first port. That made it easier for me to follow what was happening. Pushing the joystick forward caused execution to speed up. The border color on the screen changed as the speed changed to let me know when I reached the maximum speed--very nice touch. After I let the program run for a little while, I typed "B" to break the program (pushing the joystick button worked equally well). I tried typing HELP because the tutorial manual told me I could, and reread the command summaries. This time they made some sense. Typing "CONT" caused my program to resume execution, and the numbers started to march by again. "NEW" cleared out my program area.

This brought me to the end of lesson 1 (which is only a single page of the tutorial manual--so elegantly is this constructed). The self-test at the end of the chapter was fairly simple, but posed relevant questions.

Lesson 2 explained the SC's ability to single-step through execution--a great boon to understanding at one's own pace.

Lessons 3-5 discussed algebra, conditionals, and looping, all in the same straightforward manner as presented in lesson 1. Thus, the first two lessons concentrated on the mechanics of the computer and the internal representation of program information, while the last three lessons covered fundamental concepts of programming. That is exactly the way any course in assembly language should be taught.

Edu-Soft's Simulated Computer is an ideal tool for teaching children--or even adults--the basics of programming in machine or assembly language. The Simulated Computer supports ten op-codes: STOP/SKIP; LOAD; STORE; ADD; SUBTRACT; MULTIPLY; DIVIDE; INPUT; OUTPUT; and JUMP. Additionally, the SKIP instruction has six conditional variations. The eight error messages provide adequate feedback to correct an execution-time mistake.

The internal documentation fo this program (accessible via the HELP command) is wonderful, and the tutorial manual is excellent. The commands which are summarized within the program are discussed in greater detail in the manual. The assignments that are given in the self-test are worthwhile and relevant (answers are provided in the back of the manual), and additional problems are suggested for practice in the end.

The copy of the Simulated Computer that was given to me supports the joystick features that I mentioned earlier. These are not yet explained in the user manual, but will be when the package is made generally available this summer. I expect this addition to be as well-documented as is the rest of the program. It's obvious that a lot of thought went into the production of this exceptional piecd of software. Any young programmer will be able to get a lot out of it.

-K.L.

## *JUST FOR KIDS?*

## ANNIE DOOLIE

Annie Doolie is a girl who is very particular about what she likes and does not like. This program will test your or your friends' understanding of Annie. Perhaps you can tell by the program what it is all about. When you run it, the program will continue asking for the right answer until it has been given twice in a row. Feel free to make the data list longer or to change it, but be sure you to adapt the variable "try," which keeps score of the number of examples given and is set to one again after reading the last data. The program only ends after the problem has been solved; if desperate, press [SYSTEM RESET] or [BREAK].

```
10 DIM A$(1),EX1$(10),EX2$(10),ANS1$(15),ANS2$(15)
20 REM TITLE
30 GRAPHICS 2+16:POSITION 4,4:PRINT #6;"ANNIE DOOLIE"
40 FOR CL=1 TO 150:POKE 708,CL:SOUND 0,CL,10,8:FOR W=1 TO 15:NEXT W
50 NEXT CL:SOUND 0,0,0,0
55 CORRECT=0:TRY=0:PAUSE=250
60 GRAPHICS 0:A1=0:A2=0:TRY=TRY+1:IF TRY=7 THEN TRY=1:RESTORE 300
65 READ EX1$,EX2$:? :? "Annie Doolie likes ";EX1$
70 PRINT "but she doesn't like ";EX2$
80 POSITION 4,5:? "NOW IT'S YOUR TURN:"
90 PRINT :PRINT "Annie Doolie likes ";:INPUT ANS1$
100 PRINT "but she doesn't like ";:INPUT ANS2$
110 REM CHECKING THE ANSWERS
120 FOR I=2 TO LEN(ANS1$):IF ANS1$(I,I)=ANS1$(I-1,I-1) THEN A1=1
130 NEXT I
140 FOR I=2 TO LEN(ANS2$):IF ANS2$(I,I)=ANS2$(I-1,I-1) THEN A2=1
150 NEXT I
160 IF A1=0 OR A2=1 THEN ? :? "WRONG... TRY AGAIN":GOSUB PAUSE:CORRECT=0:GOTO 60
170 CORRECT=CORRECT+1:? :? " RIGHT!!!":IF CORRECT=1 THEN ? "GIVE ME ANOTHER PAIR
":GOSUB PAUSE:GOTO 60
180 ? "       AGAIN.":POSITION 10,13:? " CONGRATULATIONS ":REM IN INVERSE VIDEO
190 END
250 FOR WAIT=1 TO 800:NEXT WAIT:RETURN
300 DATA PUPPIES,DOGS
310 DATA BEEF,PORK
320 DATA GLASSES,CUPS
330 DATA COFFEE,TEA
340 DATA CHESS,CHECKERS
350 DATA BOOKS,MAGAZINES
```

# Lisp and Natural Language

by Peter Swiggart

Why is LISP such an interesting language? Why is it so useful for research into artificial intelligence? The answer to both questions is essentially the same. LISP is interesting and important because it is close to natural language and provides insight into communication situations. The nice thing about LISP is that it is linked to a language such as English in two complementary ways. LISP expressions can have a two-part syntactic structure that resembles the division of a sentence into subject and predicate, or into noun phrase and verb phrase. But a complex expression in LISP can also be what amounts to the quotation of a natural language sentence. This means that LISP can be used as a way of manipulating English expressions, on the basis of syntactic and other transformation rules built into LISP programs

Some entertaining uses can be made of this capacity. For example, Datasoft includes in its Inter-LISP package for Atari users a simplified version of the well-known Doctor program. A user is asked to type in answers to the kinds of questions a psychiatrist might be expected to ask. The LISP program searches for key words or phrases in input sentences and then applies functions which yield 'appropriate' responses and further questions from Doctor. For example, the typed-in sentence 'I am worried about X' gets the response 'Why are you worried about X?' The appearance of 'mother' (or 'father') in a list will trigger responses of the type 'Tell me more about your mother (father).' If you simply answer 'yes' or 'no' to a query from Doctor you are rebuked by 'Don't be so short with me', and the use of words like 'damn' or 'hell' draws Doctor's criticism. When the full linguistic resources of a rich LISP program are brought to bear upon an unsuspecting human 'patient' the results can be devastatingly realistic. Datasoft's version is very limited (an uncooperative user is likely to draw an early 'Sorry your time is up') but it serves as a framework that even the inexperienced programmer can readily expand to suit his research or entertainment needs.

Although expressions based on LISP functions can resemble natural sentences, there is an important difference that deserves attention. In a natural language we use names to designate objects in the world, and we think of a sentence's truth or falsity as dependent upon facts or circumstances that are independent of language. In LISP there are expressions which have arguments or return values in the manner of natural language names, and we also find expressions that yield truth values -- that are either true or false. But the values in question are always expressions in LISP; even truth and falsity, or T and NIL, are LISP expressions. That is to say, the LISP function that is applied to a given expression always returns another expression as its value. The result is a 'closed' language system in which meaning is entirely self-contained, as if we had a natural language in which words always had other words as their meaning and never referred to anything outside language.

LISP is far from unique among computer languages in having 'meanings' that are part of the language structure. But LISP is also a language that helps us understand how a language like English is more self-contained than we might think. A few psychologists and linguistics, Jean Piaget in particular, have argued that the words and sentences of a natural language only have other words and sentences as their meaning, specifically the words and sentences we use in associated contexts. This view is supported by the effectiveness of <u>Doctor</u> and related LISP programs. In language communication the interchange of word associations and syntactic choices may be as important for understanding as knowledge of what is being talked about.

It is clear that LISP programs can simulate many features of natural language conversation. This provides an opportunity for language research, but it also gives the user a chance to talk to his Atari as well as enjoy its graphics and its games.

# STRINGS IN TWO BASICS

*by Frieda Lekkerkerker*

The ability to handle strings can turn your computer into a wonderful conversation piece. When you are allowed to use your everyday English words, phrases, names and places in a program, you may feel a natural closeness to the machine, a sharing of a common language. BASIC has enough string features to interest the man or woman of many words. This article will discuss some of those features in the 8K ROM BASIC as well as in ATARI Microsoft BASIC - a great excuse to do a little comparison between the two versions.

One of the complaints about the 8K BASIC is that it has no string arrays, no (easy) opportunity to represent and manipulate a list of words and phrases. Microsoft BASIC allows up to 3-dimensional string arrays, a filing cabinet full of data. The maximum length of a string in Microsoft is 255, whereas the ROM BASIC lets you work with long strings, whose limit only depends on memory size - a powerful feature that can be used for more than just text purposes. Player data can be part of a long string and vertical motion achieved through moving the data up and down in the string; a long string can hold the bytes of a character set or those of a machine language routine , which can then be found through the ADR function. But here we are concerned with the more literal functions of strings and will leave those other tricks to a future article.

The maximum length, however, is important, because long strings will allow you to simulate string arrays, if you give it some thought. You'll find two short programs (listings 1a and 1b), examples of a bubble sort of a 2-dimensional array with a choice of sort field. The arrays hold the names, ID's and grades of five students; it is obvious that the idea can be changed and adapted to suit one's purpose. The Microsoft version is clearly easier to read and certainly easier to write. The program includes some nifty features of this new BASIC: the CLS command clears the screen and it can be followed by a number to determine the border color in Gr.0 and 8 and the background in the other modes. The TAB command obviously makes things a lot more pleasant for the programmer; AT in PRINT AT is similar to the POSITION command. In the 8K BASIC version I divided the long string into substrings of ten characters each, combining three substrings to emulate a matrix of 3 by 5 strings. It works just as well as Microsoft Basic, and, if you try it out on a long list, you will notice that the ROM BASIC runs faster; another method of sorting may give a different result.

The next two listings (2a and 2b) show the extensive string functions in Microsoft Basic and their deplored absence in the ROM version. Users of other BASICs will be familiar with them. LEFT$ and RIGHT$ let you isolate a portion of the string starting from the left or ending at the right side of the string with a specified (by the second parameter) number of characters. MID$, not used in the example, returns any portion of a string, whose beginning and end depend on the second and third parameter provided. INSTR looks for a small string within a larger one, a cumbersome task in the 8K BASIC. The ease of the INSTR function is illustrated by the two programs that search and replace a substring within a longer string. Notice also the reasonable way Microsoft concatenates - two simple +'s do the trick - by contrast to the step by step method of the 8K. The LINE INPUT statement differs from INPUT (in Microsoft!) in that it recognizes commas, etc. as part of the string and not (as with INPUT) as separators of a series of inputs.

There are a few more string functions in the new Basic for ATARI which will surprise and, who knows, delight the dedicated 8K Basic user. STRING$, for instance, returns a string consisting of a repetition of a given character, in case you need 40 stars in a row. And with DEFSTR a variable can be defined as a string, so that after a "DEFSTR A-D" all variables from A to D without a specific tag (#, % or $) will be considered as strings. I have not found a useful application for this feature and would be grateful for any suggestion.

What conclusion can be drawn from this brief comparison? The important thing, to my mind, is that most of the features of Microsoft can be simulated in the ROM version. But the convenience of the extra features in the former can save quite a bit of time and headache for the writer or the reader of a BASIC program. Microsoft BASIC has more goodies for clear and organized programming, paricularly in formatting and in Graphics. But then we would have to leave the world of strings for numbers and pixels. Which version you will prefer will quite likely be determined by more than words alone.

```
5 REM BUBBLE SORT OF 2-DIM STRING ARRAY
10 DIM FIL$(150),REC$(10),T$(30)
20 REM FILL STRING WITH BLANKS
30 FIL$(1)=" ":FIL$(150)=" ":FIL$(2)=FIL$
40 REM READ DATA AND PUT THEM IN THE FIELDS OF THE STRING
50 FOR I=1 TO 150 STEP 10:READ REC$:FIL$(I,I+9)=REC$:NEXT I
60 REM ASK FOR FIELD TO SORT
70 ? CHR$(125):POSITION 2,2:? "FIELD TO BE SORTED (1,2 OR 3)";:INPUT F:IF F<1 OR F>3 THEN 70
80 F=F*10-10:REM RELATIVE POSITION OF FIELDS WILL BE AT 0,10 OR 20
90 REM SORTING
100 B=0:REM IF B IS SET (=1) THEN A SORT HAS TAKEN PLACE; IF NOT SET (=0) NO SORT AND NO FURTHER LOOP
110 FOR I=1 TO 120 STEP 30
120 IF FIL$(I+F,I+F+9)<=FIL$(I+F+30,I+F+39) THEN 140:REM NO SORTING NEEDED
130 T$=FIL$(I,I+29):FIL$(I,I+29)=FIL$(I+30,I+59):FIL$(I+30,I+59)=T$:B=1
140 NEXT I:IF B=1 THEN 100
150 REM PRINT OUT THE SORTED FILE
155 ? CHR$(125):POSITION 2,2:? "NAME":POSITION 12,2:? "ID#":POSITION 22,2:? "GRADE":POSITION F+1,2:? "*":?
160 FOR I=1 TO 150 STEP 30
170 PRINT FIL$(I,I+9);FIL$(I+10,I+19);FIL$(I+20,I+29)
180 NEXT I
500 DATA PETER,46735,B-
510 DATA BILL,15624,B+
520 DATA KAREN,35264,A-
530 DATA KATHY,81425,C+
540 DATA NATHANIEL,79865,A
```

*(listing 1A)*

```
10 !2-DIM STRING ARRAY SORT IN MICROSOFT BASIC
20 CLS 240:PRINT "FIELD TO BE SORTED(1,2 OR 3)";:INPUT F:IF F<1 OR F>3 THEN 20
30 FOR I=1 TO 5:FOR J=1 TO 3
40 READ REC$:FIL$(I,J)=REC$
50 NEXT J,I
60 B=0
70 FOR I=1 TO 4:IF FIL$(I,F)<= FIL$(I+1,F) THEN 90
80 B=1:FOR J=1 TO 3:T$=FIL$(I,J):FIL$(I,J)=FIL$(I+1,J):FIL$(I+1,J)=T$:NEXT J
90 NEXT I:IF B=1 THEN 60
100 !PRINTING OUT OF FILE
110 CLS 248:PRINT TAB(1);"NAME";TAB(11);"ID#";TAB(21);"GRADE":PRINT AT(F*10-9,0);"*":PRINT
120 FOR I=1 TO 5:FOR J=1 TO 3
130 PRINT TAB(J*10-9);FIL$(I,J);
140 NEXT J:PRINT:NEXT I
150 DATA PETER,46735,B-
160 DATA BILL,15624,B+
170 DATA KAREN,35264,A
180 DATA KATHY,81425,C+
190 DATA NATHANIEL,72465,A+
```

*(listing 1B)*

```
10 !SEARCH AND REPLACE IN MICROSOFT BASIC
20 CLS 6:PRINT "TYPE A SENTENCE OF MAX.3 LINES"
30 LINE INPUT SENT$
40 PRINT :PRINT "SEARCH STRING";:INPUT SEARCH$
50 PRINT "REPLACE WITH";:INPUT REPLACE$
60 PLACE=INSTR(SENT$,SEARCH$):IF PLACE=0 THEN PRINT "STRING NOT FOUND":END
70 A$=LEFT$(SENT$,PLACE-1):Z$=RIGHT$(SENT$,LEN(SENT$)-PLACE-LEN(SEARCH$)+1)
80 NEWSENT$=A$+REPLACE$+Z$
90 PRINT:PRINT NEWSENT$
```

*(listing 2B)*

```
5 REM SEARCH AND REPLACE
10 DIM SENT$(100),A$(100),Z$(100),SEARCH$(30),REPLACE$(30)
20 ? CHR$(125):? "TYPE SENTENCE":INPUT SENT$
30 ? :? "SEARCH STRING";:INPUT SEARCH$
40 ? "REPLACE WITH";:INPUT REPLACE$
45 FOUND=0
50 FOR I=1 TO LEN(SENT$)-LEN(SEARCH$)+1
60 IF SENT$(I,I+LEN(SEARCH$)-1)=SEARCH$ THEN FOUND=1:GOTO 90
70 NEXT I
75 IF  NOT FOUND THEN ? :? "STRING NOT FOUND":END
80 REM SEPARATE THE TWO PARTS EXCLUDING THE SEARCH STRING
90 IF I=1 THEN A$="":GOTO 110:REM SEARCH STRING IS THE FIRST OF THE SENTENCE
100 A$=SENT$(1,I-1)
110 IF I=LEN(SENT$)-LEN(SEARCH$)+1 THEN Z$="":GOTO 130:REM SEARCH STR. IS THE LAST OF THE SENT.
120 Z$=SENT$(I+LEN(SEARCH$),LEN(SENT$))
130 SENT$(1)=" ":SENT$(100)=" ":SENT$(2)=SENT$:REM CLEAR SENTENCE W/ BLANKS FOR NEW ARRANGEMENT
140 REM NOW CONCATENATE INCLUDING THE NEW STRING
150 SENT$=A$:SENT$(LEN(SENT$)+1)=REPLACE$:SENT$(LEN(SENT$)+1)=Z$
160 ? :? SENT$
```

*(listing 2A)*

The purpose of this section is to inform users of the availability of software, hardware, and accessories for the Atari. It is not intended as a review of these products, but we will indicate with an asterisk those items that are expected to become, or are already, fast movers. Many of the products listed here will be reviewed in future issues. Some products may not be new in themselves, but may now be available in a revised version or on a new medium. These cases are indicated by underlining the new feature. Items are listed as follows: product name, manufacturer, category, medium, requirements, price.

Apple Panic
Broderbund
Arcade Game
1 Disk
Joystick

*Atari Microsoft BASIC
Atari, Inc.
Programming language
1 Disk
32K (48K recommended)
$89.95

*Atari Pac-Man
Atari, Inc.
Arcade Game
ROM cartridge
16K, joystick
$44.95

Atari Prom-It EDS
MPC Peripherals Corp.
EPROM programmer
Peripheral device
32K, Disk drive
$199.95

Chicken
Synapse Software
Arcade Game
Disk or cassette
Paddles
$29.95

*Crossfire
On-Line Systems
Arcade game
Disk or cassette
16K, joystick
$29.95

Custom Wood Desk
System Support
furniture
Oak, walnut, cherry
No system requirements
$399.95

Data Perfect
LJK
Database Manager
1 Disk
32K
$99.95

Deadline
Infocom, Inc.
Murder mystery adventure
2 Disks
40K
$49.95

*Frogger
On-Line Systems
Arcade game
1 Disk
40K, joystick
$34.95

Hockey
Gamma Software
Arcade game
Disk or cassette
16K
Joystick
$29.95

*Jawbreaker
On-Line Systems
Arcade game
Disk or cassette
16K, joystick
$29.95

K-DOS
K-Byte
Systems software
1 Disk
32K (48K recommended)
$89.95

Macro-Assembler
Atari, Inc.
Programming language
1 Disk
32K (48K recommended)
$89.95

*Megalegs
Mega Software
Arcade Game
Disk or cassette
24K or 16K, joystick
$29.95

Micro-Painter
Datasoft
Electronic coloring book
1 Disk
48K, joystick optional
$34.95

Pascal
Atari Program Exchange
Programming language
2 Disks
2 Disk drives, 32K (48K recommended)
$49.95

*Percom RFD
Percom, Inc.
Double-density disk drive
All-metal construction
32K
$799.95

Professional Joystick
Game Tech
Accessory
Metal case
N/A
$34.95

*Raster Blaster
BudgeCo
Arcade Game
1 Disk
32K, 2 joysticks
$29.95

*Threshold
On-Line Systems
Arcade game
1 Disk
40K, 1 joystick
$39.95

Tumblebugs
Datasoft
Arcade game
1 Disk
40K, joystick
$29.95

*Zork I & Zork II
Infocom, Inc.
Adventure game
1 Disk each
32K
$39.95

In this column we will present a list of reading material to aid the literature-starved Atari user. Where there is a sub-heading for a book, manual, or magazine, we are highlighting a specific article or series of articles. Otherwise, the entire publication is suggested.

## MAGAZINES

A.N.A.L.O.G. 400/800 Magazine
P.O. Box 23
Worcester, MA 01603
(617) 892-3488
Editors/Publishers: Michael Des Chenes & Lee Pappas
Frequency: Bi-monthly
Recommended for ALL users.

ANTIC
297 Missouri Street
San Francisco, CA 94107
(415) 864-0886
Editor/Publisher: James Capparell
Frequency: Bi-monthly
Recommended for ALL users.

BYTE
70 Main Street
Peterborough, NH 03458
(603) 924-9281
Editor in Chief: Christopher Morgan
Frequency: Monthly
--"The Atari Tutorial" by Chris Crawford et al. Serial since the 9/82 issue.
Recommended for INTERMEDIATE/ADVANCED users.

COMPUTE!
P.O. Box 5406
Greensboro, NC 27403
(919) 275-9809
Editor/Publisher: Robert C. Lock
Frequency: Monthly
Recommended for ALL users.

CREATIVE COMPUTING
39 East Hanover Avenue
Morris Plains, NJ 07950
(201) 540-0445
Publisher/Editor-in-Chief: David H. Ahl
Frequency: Monthly
--Columns by David & Sandy Small (since 6/81)
--Outpost Atari
Recommended for ALL users.

## BOOKS

The Atari Assembler
by Don and Kurt Inman
Reston Publishing Company, Inc.
Reston, VA: 1981
$12.95
Recommended for INTERMEDIATE users.

Atari Sound and Graphics
by Herb Moore, Judy Lower, & Bob Albrecht
John Wiley & Sons, Inc.
New York: 1982
$9.95
Recommended for ALL users.

COMPUTE!'s First Book of Atari
COMPUTE! Books
Greensboro,NC: 1981
$12.95
Recommended for ALL users.

Inside Atari DOS
Compiled by Bill Wilkinson
COMPUTE! Books
Greensboro,NC: 1982
$19.95
Recommended for ADVANCED users.

Picture This!
by David D. Thornburg
Addison-Wesley Publishing Company
$14.95
Recommended for CHILDREN.

Starting FORTH
by Leo Brodie
Prentice-Hall, Inc.
Englewood Cliffs, NJ: 1981
$15.95
Recommended for INTERMEDIATE users.

Your Atari Computer:
a Guide to Atari 400/800 Personal Computers
by Lon Poole, Martin McNiff, and Steven Cook
Osborne/McGraw-Hill
Berkeley, CA: 1982
$14.95
Recommended for ALL users.

## MANUALS

"Atari Technical User Notes"
Atari, Inc.
$27.00
Recommended for ADVANCED users.

"Atari 810 DOS Utilities Source Listing (DOS II)"
Atari, Inc.
$5.95
Recommended for ADVANCED users.

"Atari 400/800 Operating System Source Listing"
Atari, Inc.
$19.95
Recommended for ADVANCED users.

"De Re Atari"
Atari Program Exchange
$19.95
Recommended for INTERMEDIATE/ADVANCED users.

################################################################################

## BACH on the ATARI

by Richard Kaye

The ATARI Home Computer is naturally equipped in BASIC to produce music in one timbre, four voices, from the C below the middle C to the second C above the middle C ("high C"). This corresponds roughly to the style of writing used by Bach in his harmonizations of Lutheran chorales and is similar to most hymn writing for 4-part choir, where harmonic, 4-part writing is used. There are limitations to the program presented:

1. It does not go low enough. The bass line should go down to G or F. We could Poke 53768,1 after the sound statements on line 30 to lower the pitch, but, unfortunately, this will lower all four voices <u>and</u> by the unmusical factor of 0.2344. I'll work this out at a later date for the bass alone, but for now we are limited to the "C" on p. 58 fo the "Basic Reference Manual." 255 will sound, but it is not quite a true "B," so you can't use it for music.
2. Articulation affects all four voices.
3. Volume affects all four voices.
4. The repetition method given provides only for an exact repeat of the starting section, without primo and secondo endings. Because, in fact, this is what Bach does in the Chorales, the method works just fine for them.

To use this program for your own choice of music you must supply data statements from line 90 to 499 (you can skip up to some higher, unused number for more data if that isn't enough). Also the timing number "200" in line 40 is only a guess. You have to adjust it for each selection.

First find a piece of music in four or fewer parts; homophonic is nice, but not absolutely necessary. However, you can't do four against 3 or 2 against 3 or 7 against 5 or any exotic syncopation without driving yourself nutty, and turns, trills, and grace notes are impractical, so forget 'em. It will be nice if the music does not go below C ( the one below middle "C," known as "c." The program listing uses Chorale #46 as an example.

```
1 REM THIS IS A PROGRAM TO PLAY MUSIC
2 REM ON THE ATARI 400/800. 4 PART HARMONY,
3 REM SUCH AS THE BACH CHORALE HARMONIZATIONS IS ASSU
MED.
4 REM FURTHER INSTRUCTIONS ARE AT LINE 75
5 N=0:L=0:PRINT "}":GOSUB 1000
10 READ A,B,C,D
15 IF D>258 THEN 700
20 IF D=256 THEN GRAPHICS 0:PRINT "}":END
25 IF D=257 THEN 500
27 IF D=258 THEN 600
30 SOUND 0,A,10,L:SOUND 1,B,10,L-3:SOUND 2,C,10,L-3:S
OUND 3,D,10,L-2
40 FOR W=1 TO 200:NEXT W
60 GOTO 10
70 END
75 REM THERE ARE FOUR PARTS TO THE HARMONY.
76 REM A=SOPRANO(HIGHEST VOICE)
77 REM B=ALTO(UPPER MIDDLE VOICE)
78 REM C=TENOR(LOWER MIDDLE VOICE)
79 REM D=BASS(LOWEST VOICE)
80 REM SUPPLY DATA STATEMENTS FOR ABCD
81 REM AT A DURATION EQUAL TO THE SHORTEST NOTE IN YO
UR SELECTION.
82 REM TO END,LAST DATA SHOULD BE 0,0,0,256.
83 REM TO ARTICULATE BETWEEN TWO NOTES OF THE SAME PI
TCH IN VOICE A
84 REM INSERT DATA 0,0,0,257.
85 REM TO REPEAT ONCE FROM BEGINNING
86 REM INSERT DATA 0,0,0,258.
87 REM YOU MAY VARY TEMPO BY CHANGING FIGURE IN LINE
40;
88 REM VOLUME IS CONTROLLED BY INSERTING A FIGURE FRO
M 262 TO 268 INTO THE DATA FOR VARIABLE "D."
89 REM START YOUR DATA AT LINE 90. DO NOT GO BEYOND L
INE 499.
100 DATA 0,0,0,266
101 DATA 53,72,85,217,53,72,85,217
102 DATA 57,72,96,144,57,76,96,144,64,85,108,144,64,8
5,108,153,57,96,114,144,57,96,114,162,72,85,108,173,7
2,96,114
103 DATA 173,64,108,128,162,64,108,144,173,57,96,162,
193,57,96,162,193
104 DATA 53,85,144,217,53,85,144,217,53,85,144,217,0,
0,0,257,53,85,144,217,53,81,144,193
105 DATA 0,0,0,257,53,72,108,173,53,81,108,173,72,85,
108,173,72,85,108,162
106 DATA 0,0,0,257,72,96,114,144,72,96,114,128,85,108
,121,144,81,108,128,162
107 DATA 72,108,144,173,72,108,144,193,81,96,144,217,
81,96,144,230
108 DATA 85,108,144,217,85,108,144,217,85,108,144,217
,0,0,0,257,85,108,144,217,85,96,144,230
109 DATA 64,85,144,204,64,85,144,255,0,0,0,257,64,96,
153,193,64,96,153,217
110 DATA 72,96,144,230,64,76,108,255,57,72,114,144,57
,72,128,153
111 DATA 53,72,144,173,57,72,114,193,64,72,85,217,64,
76,96,193
112 DATA 72,96,114,144,72,96,114,144,72,96,114,144,53
,85,108,144,53,96,128,153
113 DATA 57,96,128,144,57,85,144,173,64,85,144,217,64
,96,162,162
114 DATA 72,114,144,162,72,108,144,173,64,128,162,162
,72,114,144,173
115 DATA 81,108,128,217,85,108,162,217,96,108,144,144
,96,114,162,144,108,144,173,217,108,144,173,218,108,1
44,173,2
116 DATA 0,0,0,256
500 SOUND 0,A,10,0:SOUND 1,B,10,0:SOUND 2,C,10,0:SOUN
D 3,D,10,0
510 FOR I=1 TO 2:NEXT I
520 GOTO 10
600 SOUND 0,A,10,0:SOUND 1,B,10,0:SOUND 2,C,10,0:SOUN
D 3,D,10,0
610 FOR J=1 TO 3:NEXT J
620 N=N+1
630 IF N>1 THEN 10
640 RESTORE :GOTO 10
700 SOUND 0,A,10,0:SOUND 1,B,10,0:SOUND 2,C,10,0:SOUN
D 3,D,10,0
710 L=D-258
720 GOTO 10
1000 GRAPHICS 2:POSITION 5,3
1010 PRINT #6;"J.S.BACH"
1020 ? "CHORALE HARMONIZATION #46"
1030 ? "VOM HIMMEL HOCH, DA KOMM' ICH HER"
1040 POKE 755,1:RETURN
```

# Player Movement in Assembly

by Karen Leavitt

This is a template program that enables the user to create a player in assembler language and move it around using a joystick. The equates in the beginning assign labels to important locations in memory that are necessary to access in order to enable and use Player/Missile graphics. P0LO and P0HI are two page zero locations used as pointers to the beginning of Player RAM.

The first thing the program does is to set up Player RAM eight pages below the top of memory. It does this by subtracting eight from the contents of location 106 (free pages addr) and stores this new number at PMBASE. Next, we poke the graphics control register (GRACTL) with a 3 to enable P/M DMA. Poking a 46 into DMACTL sets up 2-line resolution for the players. The player is then defined in DEFLOOP by grabbing a byte from TABLE and sticking it into Player RAM (I made my player 8 bytes long; you can change the length by adding bytes to the TABLE, and changing the LENGTH value).

At JOYSTICK, the value of the joystick port is read and appropriate routines are executed to move the player in the proper direction. Then a delay loop is executed so you are able to see the player moving across the screen (otherwise it strobes because of the speed of machine language).

You can experiment by adding a second player, or using some missiles as well. I've tried to make this as general as possible by using equates and defining variables in .BYTE statements. I've published the assembler listing for this program instead of the source listing for purposes of legibility and to enable BASIC programmers to poke the hexadecimal values into their own BASIC programs (after converting to decimal, of course). If any user has a good variation on this program, or if there are any questions regarding pieces of code, please contact me c/othis journal.

```
0000                  *=     $2000
D407           PMBASE =      54279
022F           DMACTL =      559
D01D           GRACTL =      53277
D000           HPOS0  =      53248
02C0           COLP0  =      704
D300           STICK0 =      $D300
00CB           P0LO   =      $CB
00CC           P0HI   =      $CC
2000 A56A             LDA    106        ;LOAD HIGHEST PAGE NUMBER
2002 E908             SBC    #8         ;SUBTRACT 8 (PAGES) TO MAKE ROOM FOR PLAYER/MISSILE AREA
2004 8D07D4           STA    PMBASE     ;STORE BEGINNING OF P/M AREA AT PMBASE
2007 6902             ADC    #2         ;ADD TWO PAGES (512 BYTES)
2009 85CC             STA    P0HI       ;THIS IS HIGH BYTE OF PLAYER 0 ADDRESS
200B A903             LDA    #3
200D 8D1DD0           STA    GRACTL     ;ENABLE P/M DMA  BY POKING A 3 INTO GRACTL
2010 A92E             LDA    #46
2012 8D2F02           STA    DMACTL     ;ENABLE PLAYFIELD DMA AND SET 2-LINE P/M RESOLUTION
2015 AD0320           LDA    COLOR
2018 8DC002           STA    COLP0      ;POKE DESIRED COLOR (I CHOSE PINK) INTO COLOR REGISTER
```

```
201B        0295        .PAGE
            0300 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            0310 ;x HERE'S WHERE WE DEFINE THE PLAYER x
            0320 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
201B A200   0330        LDX  #0        ;X IS INDEX TO TABLE CONTAINING DEFINITION BYTES
201D ACD020 0340        LDY  YPOS      ;Y INDEXES LOCATION IN PLAYER RAM--START FROM OFFSET YPOS
2020 BDD420 0350 DEFLOOP LDA TABLE,X   ;LOAD A BYTE FROM DEFINITION TABLE
2023 91CB   0360        STA  (POLO),Y  ;STORE IT IN PLAYER RAM
2025 C8     0370        INY            ;INCREMENT
2026 E8     0380        INX            ;POINTERS
2027 ECD120 0390        CPX  LENGTH    ;MAKE SURE WE DON'T EXCEED SPECIFIED LENGTH
202A D0F4   0400        BNE  DEFLOOP   ;CONTINUE TO GRAB AND POKE
202C AECF20 0410        LDX  XPOS      ;GET ARBITRARY HORIZONTAL POSITION FROM LIST
202F 8E00D0 0420        STX  HPOS0     ;STORE IT IN HORIZ. POS. REGISTER
            0430 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            0440 ;x HERE'S WHERE WE READ THE JOYSTICK x
            0450 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2032 AE00D3 0460 JOYSTICK LDX STICK0   ; READ JOYSTICK
2035 8ED220 0470        STX  STICKNUM  ;SAVE JOYSTICK VALUE
2038 E0FF   0480        CPX  #$FF      ; IS JOYSTICK IN UPRIGHT POSITION?
203A F0F6   0490        BEQ  JOYSTICK  ; YES, GO READ IT AGAIN
203C E0F8   0500        CPX  #$F8
203E 300C   0510        BMI  RIGHT     ;IF STICK<8 MOVE RIGHT
2040 E0FC   0520        CPX  #$FC
2042 3022   0530        BMI  LEFT      ;IF STICK<12 MOVE LEFT
2044 E0FE   0540        CPX  #$FE      ;IS STICK=14?
2046 F038   0550        BEQ  UP        ;YES, MOVE UP
2048 E0FD   0560        CPX  #$FD      ;IS STICK=13?
204A F053   0570        BEQ  DOWN      ;YES, MOVE DOWN
            0590 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            0600 ;x HERE'S WHERE WE MOVE RIGHT x
            0610 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
204C ADCF20 0620 RIGHT  LDA  XPOS
204F 18     0630        CLC
2050 6901   0640        ADC  #1        ;INCREMENT HORIZONTAL POSITION BY ONE
2052 8DCF20 0650        STA  XPOS      ;STORE NEW POSITION BACK IN XPOS
2055 8D00D0 0660        STA  HPOS0     ;POKE NEW POSITION INTO HORIZ. POS. REGISTER
2058 AED220 0670        LDX  STICKNUM  ;CHECK TO SEE IF THIS WAS A DIAGONAL MOVE
205B E0F6   0680        CPX  #$F6      ;UP AND RIGHT?
205D F021   0690        BEQ  UP
205F E0F5   0700        CPX  #$F5      ;DOWN AND RIGHT?
2061 F03C   0710        BEQ  DOWN
2063 4CC220 0720        JMP  DELAY
            0730 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            0740 ;x HERE'S WHERE WE MOVE LEFT  x
            0750 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2066 ADCF20 0760 LEFT   LDA  XPOS      ;GET CURRENT HORIZONTAL POSITION
2069 38     0770        SEC
206A E901   0780        SBC  #1        ;DECREMENT IT
206C 8DCF20 0790        STA  XPOS      ;STORE IT BACK IN XPOS
206F 8D00D0 0800        STA  HPOS0     ;STORE IT IN HORIZ. POS. REGISTER
2072 AED220 0810        LDX  STICKNUM  ;CHECK FOR DIAGONAL
2075 E0FA   0820        CPX  #$FA      ;UP AND LEFT?
2077 F007   0830        BEQ  UP
2079 E0F9   0840        CPX  #$F9      ;DOWN AND LEFT?
207B F022   0850        BEQ  DOWN
207D 4CC220 0860        JMP  DELAY
```

```
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 ;x HERE'S WHERE WE MOVE UP    x
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2080 AED120      UP     LDX  LENGTH    ;GET LENGTH OF PLAYER IN X
2083 ACD020             LDY  YPOS      ;Y REGISTER CONTAINS Y OFFSET (TO BE DECREMENTED)
2086 B1CB        UPLOOP LDA  (POLO),Y  ;GRAB FIRST BYTE OF PLAYER
2088 88                 DEY            ;MOVE POINTER UP ONE TO STORE THIS BYTE ONE HIGHER
2089 91CB               STA  (POLO),Y  ;STORE BYTE AT NEW, HIGHER OFFSET
208B C8                 INY
208C C8                 INY            ;MOVE Y TO POINT TO NEXT BYTE TO BE MOVED
208D CA                 DEX            ;DECREMENT LENGTH COUNTER
208E D0F6               BNE  UPLOOP    ;CONTINUE MOVING UNTIL LENGTH EXHAUSTED
2090 A900               LDA  #0
2092 88                 DEY
2093 91CB               STA  (POLO),Y  ;ZERO OUT LAST BYTE OF PLAYER TO ERASE LAST LINE
2095 ACD020             LDY  YPOS
2098 88                 DEY
2099 8CD020             STY  YPOS      ;DECREMENT Y OFFSET POINTER
209C 4CC220             JMP  DELAY
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 ;x HERE'S WHERE WE MOVE DOWN x
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxx
209F AED120      DOWN   LDX  LENGTH
20A2 E8                 INX            ;GET THE LENGTH OF PLAYER +1 IN REGISTER X
20A3 ADD020             LDA  YPOS
20A6 6DD120             ADC  LENGTH
20A9 E901               SBC  #1        ;ACCUMULATOR HAS Y OFFSET+LENGTH-1
20AB A8                 TAY            ;PUT THIS NUMBER INTO Y REG.
20AC B1CB        DOWNLOOP LDA (POLO),Y ;GRAB A BYTE OF THE PLAYER
20AE C8                 INY            ;INCREMENT Y TO POINT TO NEXT BYTE
20AF 91CB               STA  (POLO),Y  ;STORE BYTE AT NEW LOCATION IN PLAYER RAM
20B1 88                 DEY
20B2 88                 DEY            ;POINT TO NEXT BYTE TO MOVE
20B3 CA                 DEX
20B4 D0F6               BNE  DOWNLOOP
20B6 C8                 INY
20B7 A900               LDA  #0
20B9 91CB               STA  (POLO),Y  ;ZERO OUT TOP BYTE OF PLAYER TO PREVENT LEAVING TRAIL
20BB ACD020             LDY  YPOS
20BE C8                 INY            ;INCREMENT Y OFFSET
20BF 8CD020             STY  YPOS      ;STORE NEW OFFSET AT YPOS
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 ;x HERE WE ADD A DELAY TO THE PLAYER MOVEMENT x
                 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
20C2 A202        DELAY  LDX  #2
20C4 A0FF        YLOOP  LDY  #255
20C6 88                 DEY
20C7 D0FB               BNE  YLOOP
20C9 CA                 DEX
20CA D0F6               BNE  DELAY
20CC 4C3220             JMP  JOYSTICK
20CF 78          XPOS   .BYTE 120
20D0 40          YPOS   .BYTE 64
20D1 08          LENGTH .BYTE 8
20D2 00          STICKNUM .BYTE 0
20D3 58          COLOR  .BYTE 88
20D4 FF          TABLE  .BYTE 255,255,255,255,255,255,255,255
20D5 FF
20D6 FF
20D7 FF
20D8 FF
20D9 FF
20DA FF
20DB FF
```

Chart of Color Values in SETCOLOR instruction

| NUMBER | COLOR |
|---|---|
| 0 | Gray |
| 1 | Light orange (gold) |
| 2 | Orange |
| 3 | Red-orange |
| 4 | Red |
| 5 | Pink |
| 6 | Purple-blue |
| 7 | Dark blue |
| 8 | Blue |
| 9 | Light blue |
| 10 | Turquoise |
| 11 | Blue-green |
| 12 | Green |
| 13 | Yellow-green |
| 14 | Orange-green |
| 15 | Light orange |

figure 1

COLOR/SETCOLOR Register Correspondence

| MODE | SETCOLOR REGISTER | COLOR REGISTER | COMMENT |
|---|---|---|---|
| 0 | 0 | N/A | |
| | 1 | N/A | Determines luminance of characters |
| | 2 | N/A | Background color |
| | 3 | N/A | |
| | 4 | N/A | Border color |
| 1 and 2 | 0 | N/A | Character color |
| | 1 | N/A | Character color |
| | 2 | N/A | Character color |
| | 3 | N/A | Character color |
| | 4 | N/A | Background color |
| 3,5, and 7 | 0 | 1 | Pixel color |
| | 1 | 2 | Pixel color |
| | 2 | 3 | Pixel color |
| | 3 | N/A | Unused |
| | 4 | 0 | Background color |
| 4 and 6 | 0 | 1 | Pixel color |
| | 1 | N/A | Unused |
| | 2 | N/A | Unused |
| | 3 | N/A | Unused |
| | 4 | 0 | Background color |
| 8 | 0 | N/A | Unused |
| | 1 | 1 | Pixel luminance |
| | 2 | 2 | Background color, pixel color |
| | 3 | N/A | Unused |
| | 4 | N/A | Border color |

(See also chart on page 53 of Atari BASIC Reference Manual)

Figure 2

# GET READY: THE TANK IS COMING...

*by Dave Hallowell*

*[If you are tired of the "READY" sign in your Basic, try this animation and amuse friend and foe. It incorporates many of the Atari Graphics features, e.g. custom character set (lines 990-1100), Player-Missile (120-150) and fine vertical scrolling (510-520).* Ed.]

```
10 REM ****        TANK          ****
20 REM ****   By Dave Hallowell   ****
30 REM ********************************
40 PAUSE=50:SILENCE=60:GOTO 70
50 FOR I=1 TO T:NEXT I:RETURN
60 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0:SOUND 3
,0,0,0:RETURN
70 POKE 752,1:? CHR$(125)
80 SOUND 1,244,6,4:SOUND 0,245,6,4:POKE 709,12
90 GOSUB 990:POKE 756,CHPAGE:REM Get redefined chara
cters
100 POKE 752,1:POSITION 10,10:? "                   "
110 POSITION 2,1:? "READY":POKE 752,1:? " "
120 A=PEEK(106)-8:POKE 54279,A:PM=256*A-39:POKE 559,
46:POKE 53277,3
130 POKE 53256,1:HPOS=53248:REM setup saucer graphic
s
140 FOR Z=PM+512 TO PM+660:POKE Z,0:NEXT Z
150 RESTORE 770:FOR I=0 TO 6:READ BYTE:POKE PM+570+I
,BYTE:NEXT I
160 DIM T$(2),Z$(2),R$(8),S$(1)
170 DL=PEEK(560)+256*PEEK(561)
180 T$="#$":R$="READY":S$=CHR$(160)
190 INDEX=5:SOUND 2,246,6,4:GOTO 270
200 REM *** FALLING LETTER SUBROUTINE ***
210 SOUND 0,90,12,15:FOR J=1 TO 6:NEXT J:GOSUB SILEN
CE
220 POSITION INDEX+1,P:? " ":FOR P=2 TO 21
230 POSITION INDEX+1,P:? Z$:T=5:GOSUB PAUSE
240 POSITION INDEX+1,P:? " ":NEXT P:IF Z$="R" THEN 6
00
250 POSITION INDEX+1,P:? "%"
260 SOUND 0,180,12,10:FOR T=1 TO 9:NEXT T:GOSUB SILE
NCE:RETURN
270 FOR D=36 TO 28 STEP -1:POSITION D,22:? T$:T=20:G
OSUB PAUSE
280 POSITION D,22:? "  ":NEXT D:POSITION INDEX+22,22
:? T$:POSITION INDEX+24,22:? "  "
290 POSITION INDEX+22,22:? T$:POSITION INDEX+24,22:?
 "  "
300 REM ***** FIRE SHOTS ******
310 FOR V=15 TO 0 STEP -1:SOUND 0,20,0,V:NEXT V:FOR
X=20 TO 1 STEP -1
320 J=X+INDEX:POSITION J,X:? ".":T=2:GOSUB PAUSE
330 POSITION J,X:? " ":NEXT X:Z$=R$(INDEX,INDEX)
340 GOSUB 210:IF Z$="R" THEN 540
350 INDEX=INDEX-1:IF LOOP=0 AND INDEX=1 THEN 380
360 GOTO 290
370 REM **** SHOOT AT   *****
380 POSITION INDEX+22,22:? T$:POSITION INDEX+24,22:?
 "  "
390 LOOP=LOOP+1:FOR V=15 TO 0 STEP -1:SOUND 0,20,0,V
:NEXT V
400 FOR X=20 TO 2 STEP -1
410 J=X+INDEX:POSITION J,X:? "."
420 POSITION J,X:? " "
430 NEXT X
440 SOUND 0,90,12,8:FOR T=1 TO 3:NEXT T
450 IF LOOP=1 THEN S$="("
460 POSITION 2,2:? S$:GOSUB SILENCE:T=40:GOSUB PAUSE
470 IF LOOP=1 THEN S$=")"
480 IF LOOP=2 THEN S$="+"
490 IF LOOP=3 THEN S$=CHR$(32)
500 IF LOOP<4 THEN GOTO 390
510 POKE DL+6,34
520 FOR V=1 TO 7:POKE 54277,V:FOR T=1 TO 20:NEXT T:N
EXT V:SOUND 0,90,12,8:FOR E=1 TO 5:NEXT E:GOSUB SILE
NCE
530 Z$="R":GOTO 540
540 FOR P=INDEX+22 TO 4 STEP -1:POSITION P,22:? T$
550 FOR T=1 TO 10:NEXT T:POSITION P,22:? "  ":IF P<8
 THEN 580
560 NEXT P
570 POSITION 2,22:? T$:POSITION 4,22:? "  ":GOSUB 21
0
580 FOR S=200 TO 90 STEP -15:SOUND 3,S,10,6:SOUND 2,
S-1,10,6:IF P>2 THEN POSITION P-1,22:? T$
590 NEXT S:GOSUB SILENCE:GOTO 560
600 FOR P=2 TO 14
```

```
610 SOUND 0,40*P/5,12,10:SOUND 1,(40*P/5)+1,12,10
620 POSITION P,22:? T$:T=P:GOSUB PAUSE
630 POSITION P,22:? "  "
640 SOUND 0,130,8,6:SOUND 1,131,8,6:GOSUB SILENCE
650 NEXT P:POSITION P,22:? T$:V=15:FOR T=1 TO 5:SOUN
D 0,200,0,V:SOUND 1,190,0,V
660 POKE 710,12:FOR K=1 TO 3:NEXT K:POKE 710,148:NEX
T T
670 FOR J=10 TO 160 STEP 5:SOUND 0,220,0,150/J:SOUND
 1,240,0,150/J:NEXT J:GOTO 780
680 GOSUB SILENCE:? CHR$(125):POKE 752,0
690 REM **** DATA FOR REDEFINED CHARS
700 DATA 64,32,19,15,63,255,127,63
710 DATA 0,0,128,224,252,255,254,252
720 DATA 0,0,0,0,0,0,0,126
730 DATA 0,126,126,126,114,112,112,0
740 DATA 0,120,112,96,96,64,0,0
750 DATA 0,112,96,0,0,0,0,0
760 REM **** DATA FOR PLAYER
770 DATA 8,28,62,127,62,28,8
780 REM **** SAUCER *******
790 FOR H=220 TO 102 STEP -2:R=INT(RND(0)*255):POKE
704,R
800 SOUND 0,R,10,6:SOUND 1,R+1,10,6:POKE HPOS,H:NEXT
 H:GOSUB SILENCE
810 REM * LIFT LETTERS
820 POKE 704,254
830 FOR J=PM+576 TO PM+657:POKE J,127:SOUND 0,12,100
,6:NEXT J
840 POKE 709,0:GOSUB SILENCE:FOR I=1 TO 5:FOR H=21 T
O 2 STEP -1
850 SOUND 0,10*H,10,4:SOUND 1,(10*H)+1,10,4
860 POSITION 15,H:? R$(I,I):FOR T=1 TO 2:NEXT T
870 POSITION 15,H:? " "
880 SOUND 0,100,12,15:SOUND 1,101,12,15:NEXT H:NEXT
I
890 FOR J=PM+660 TO PM+576 STEP -1:POKE J,0:NEXT J
900 GOSUB SILENCE:POKE 709,10
910 FOR U=102 TO 1 STEP -1:POKE HPOS,U:R=INT(RND(0)*
255):POKE 704,R
920 SOUND 0,R,10,U/15:SOUND 1,R+1,10,U/15:SOUND 2,R+
2,10,U/15
930 IF U=53 THEN POSITION 2,2:? "READY":SOUND 0,100,
12,15:FOR Z=1 TO 4:NEXT Z
940 NEXT U:GOSUB SILENCE:POKE 709,10
950 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0:POKE 7
09,10
960 SOUND 0,90,12,15:FOR J=1 TO 6:NEXT J:GOSUB SILEN
CE
970 ? " ":POSITION 13,22:? "      "
980 GOTO 980
990 REM * Redefines Char.Set, returns CHPAGE
1000 MEMTOP=PEEK(106):CHPAGE=MEMTOP-4:POKE 106,CHPAG
E:GRAPHICS 0
1010 POKE 710,0:POKE 752,1:POSITION 10,10:? " THE  T
ANK ..."
1020 CHROM=PEEK(756)*256
1030 CHRAM=PEEK(106)*256
1040 FOR N=0 TO 475:POKE CHRAM+N,PEEK(CHROM+N):NEXT
N
1050 POKE 752,1:POSITION 10,10:? "   IS COMING"
1060 RESTORE 700:FOR Z=24 TO 47:READ BYTE:POKE CHRAM
+Z,BYTE:NEXT Z
1070 FOR Z=64 TO 79:READ BYTE:POKE CHRAM+Z,BYTE:NEXT
 Z
1080 FOR Z=88 TO 95:READ BYTE:POKE CHRAM+Z,BYTE:NEXT
 Z
1090 POKE 710,146
1100 RETURN
```

In the next issues!

Reviews of
- Zork I
- Scitor Personal Finance and Record Keeping

Percom Disk Drives
GTIA Graphics Modes
More on Lisp
Music Utility
Microsoft Basic
and much more...

# PAC•MAN COMES HOME AT LAST.

Good news for Pac-Man fans. One of your favorite arcade characters is now available in a great new home computer version from ATARI.

Designed for play on ATARI 400 and ATARI 800 Home Computers, Pac-Man has sensational graphics and sound effects, and better-than-ever on-screen action.

If you're hungry for a little excitement, why not bring Pac-Man home tonight?

ATARI

**ATARI HOME COMPUTERS**

We've Brought The Computer Age Home.™

©1982 ATARI, INC. ATARI 400 and ATARI 800 are trademarks of ATARI, Inc. All rights reserved.
*PAC-MAN and characters are trademarks of Bally Midway Mfg. Co. licensed by Namco America Inc.

Available now at:

**The Bit Bucket**

1355 Washington Street • West Newton, Massachusetts 02165 • (617)964-3080